

Отчёт по лабораторной работе №10

**Программирование в командном процессоре ОС UNIX. Командные
файлы**

Щербак Маргарита Романовна

2022

1 Цель работы:

Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы.

1.1 Задание:

1. Написать скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. Способ использования команд архивации необходимо узнать, изучив справку.
2. Написать пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов.
3. Написать командный файл — аналог команды ls (без использования самой этой команды и команды dir). Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога.
4. Написать командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки.

2 Теоретическое введение:

Командные процессоры (оболочки)

Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера.

В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек:

- оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;
- C-оболочка (или csh) — надстройка на оболочке Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд;
- оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна;
- BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation).

POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ.

Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux-подобных операционных систем и переносимости прикладных программ на

уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна.

Рассмотрим основные элементы программирования в оболочке `bash`. В других оболочках большинство команд будет совпадать с описанными ниже.

Оболочка `bash` поддерживает встроенные арифметические функции. Команда `let` является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение — это единичный терм (`term`), обычно целочисленный.

Команда `let` берет два операнда и присваивает их переменной. Положительным моментом команды `let` можно считать то, что для идентификации переменной ей не нужен знак доллара; вы можете писать команды типа `let sum=x+7`, и `let` будет искать переменную `x` и добавлять к ней 7.

Команда `let` также расширяет другие выражения `let`, если они заключены в двойные круглые скобки. Таким способом вы можете создавать довольно сложные выражения.

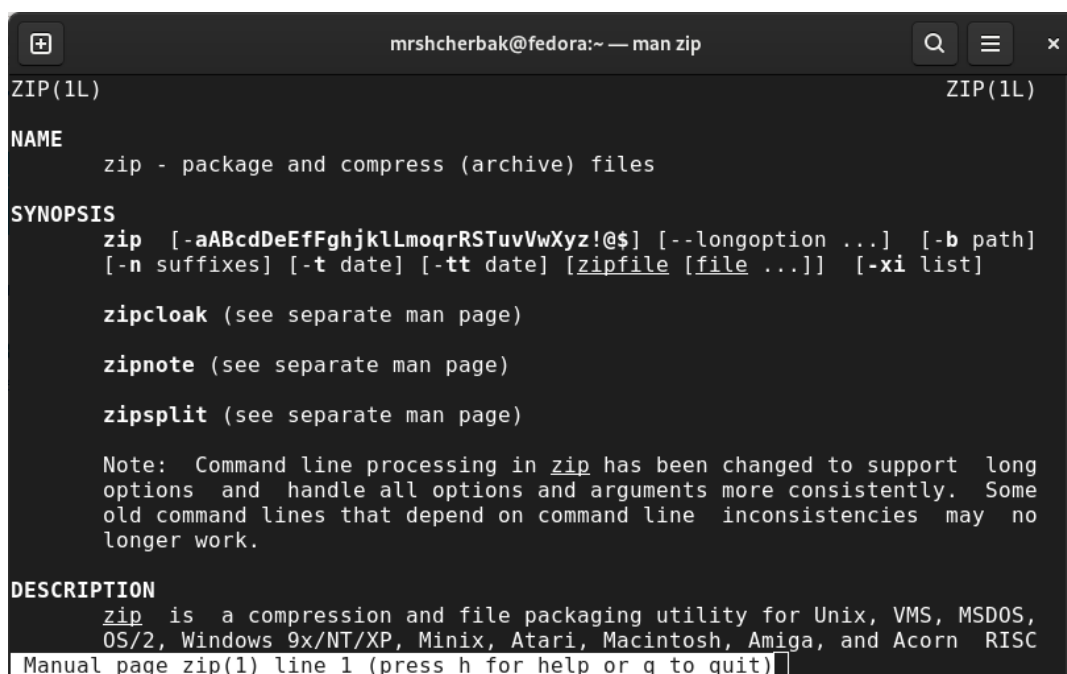
Команда `let` не ограничена простыми арифметическими выражениями.

3 Выполнение лабораторной работы:

1. Я изучила способ использования команд архивации, используя команды “man zip”, “man bzip2”, “man tar”. (Рис. 3.1 - Рис. 3.4).

```
[mrshcherbak@fedora ~]$ man zip
[mrshcherbak@fedora ~]$ man bzip2
[mrshcherbak@fedora ~]$ man tar
[mrshcherbak@fedora ~]$
```

Рис. 3.1: Команды справок по способам архивации



The screenshot shows a terminal window titled "mrshcherbak@fedora:~ — man zip". The content is the man page for the 'zip' utility. It includes sections for NAME, SYNOPSIS, and DESCRIPTION. The NAME section states that 'zip' is used to package and compress files. The SYNOPSIS section lists various command-line options like -a, -b, -n, -t, -tt, -x, etc. The DESCRIPTION section explains that 'zip' is a compression and file packaging utility for various operating systems.

```
ZIP(1L) ZIP(1L)
NAME
  zip - package and compress (archive) files

SYNOPSIS
  zip [-aABcdDeEfFghjklLmoqrRSTuvVwXyz!@$] [--longoption ...] [-b path]
  [-n suffixes] [-t date] [-tt date] [zipfile [file ...]] [-xi list]

  zipcloak (see separate man page)
  zipnote (see separate man page)
  zipsplit (see separate man page)

Note: Command line processing in zip has been changed to support long
options and handle all options and arguments more consistently. Some
old command lines that depend on command line inconsistencies may no
longer work.

DESCRIPTION
  zip is a compression and file packaging utility for Unix, VMS, MSDOS,
  OS/2, Windows 9x/NT/XP, Minix, Atari, Macintosh, Amiga, and Acorn RISC
Manual page zip(1) line 1 (press h for help or q to quit)
```

Рис. 3.2: man zip

```
mrshcherbak@fedora:~ — man bzip2
bzip2(1)          General Commands Manual          bzip2(1)

NAME
  bzip2, bunzip2 - a block-sorting file compressor, v1.0.8
  bzip2recover - recovers data from damaged bzip2 files

SYNOPSIS
  bzip2 [ -cdfkqstvl123456789 ] [ filenames ... ]
  bunzip2 [ -fkvsVL ] [ filenames ... ]
  bzip2recover [ -s ] [ filenames ... ]
  bzip2recover filename

DESCRIPTION
  bzip2 compresses files using the Burrows-Wheeler block sorting text
  compression algorithm, and Huffman coding. Compression is generally
  considerably better than that achieved by more conventional
  LZ77/LZ78-based compressors, and approaches the performance of the PPM
  family of statistical compressors.

  The command-line options are deliberately very similar to those of GNU
  gzip, but they are not identical.

Manual page bzip2(1) line 1 (press h for help or q to quit)
```

Рис. 3.3: man bzip2

```
mrshcherbak@fedora:~ — man tar
TAR(1)          GNU TAR Manual          TAR(1)

NAME
  tar - an archiving utility

SYNOPSIS
  Traditional usage
    tar {A|c|d|r|t|u|x}[GnSkUW0mpsMBiajJzZhPlRvwo] [ARG...]

  UNIX-style usage
    tar -A [OPTIONS] ARCHIVE ARCHIVE
    tar -c [-f ARCHIVE] [OPTIONS] [FILE...]
    tar -d [-f ARCHIVE] [OPTIONS] [FILE...]
    tar -t [-f ARCHIVE] [OPTIONS] [MEMBER...]
    tar -r [-f ARCHIVE] [OPTIONS] [FILE...]
    tar -u [-f ARCHIVE] [OPTIONS] [FILE...]
    tar -x [-f ARCHIVE] [OPTIONS] [MEMBER...]

Manual page tar(1) line 1 (press h for help or q to quit)
```

Рис. 3.4: man tar

2. **Первое задание.** Создала файл backup.sh, в котором писала скрипт по первому заданию. Открыла его в редакторе emacs (C-x C-f). (Рис. 3.5 - Рис.

3.7).

- Создала каталог backup в домашнем каталоге
 - Проверила его наличие с помощью команды ls
 - Предоставила право доступа на выполнение файлу backup.sh
 - Проверила файл на исполнение
- Использовала архиватор bzip2.

```
[mrshcherbak@fedora ~]$ touch backup.sh
[mrshcherbak@fedora ~]$ emacs &
[1] 2889
[mrshcherbak@fedora ~]$ mkdir backup
[mrshcherbak@fedora ~]$ ls
abcl      feathers      file14.txt~   play        Документы
acd.txt    '#file11.txt#' file.txt      reports     Загрузки
adc.cpp    file11.txt    Lab007       ski.plases  Изображения
adc.txt    file11.txt~   lab07.sh     text.cpp    Музыка
australia  file12.txt    lab07.sh~    text.txt    Общедоступные
backup     file12.txt~   may          vvvvvvvvvv 'Рабочий стол'
backup.sh  file13.txt    monthly      week3.txt   Шаблоны
bin        file13.txt~   my_os        work
conf.txt   file14.txt    otchet.txt   work
                                Видео
```

```
[mrshcherbak@fedora ~]$ chmod +x backup.sh
[mrshcherbak@fedora ~]$ ./backup.sh
[mrshcherbak@fedora ~]$ ./backup.sh
Выполнено!
```

Рис. 3.5: Создание файла и проверка на работоспособность

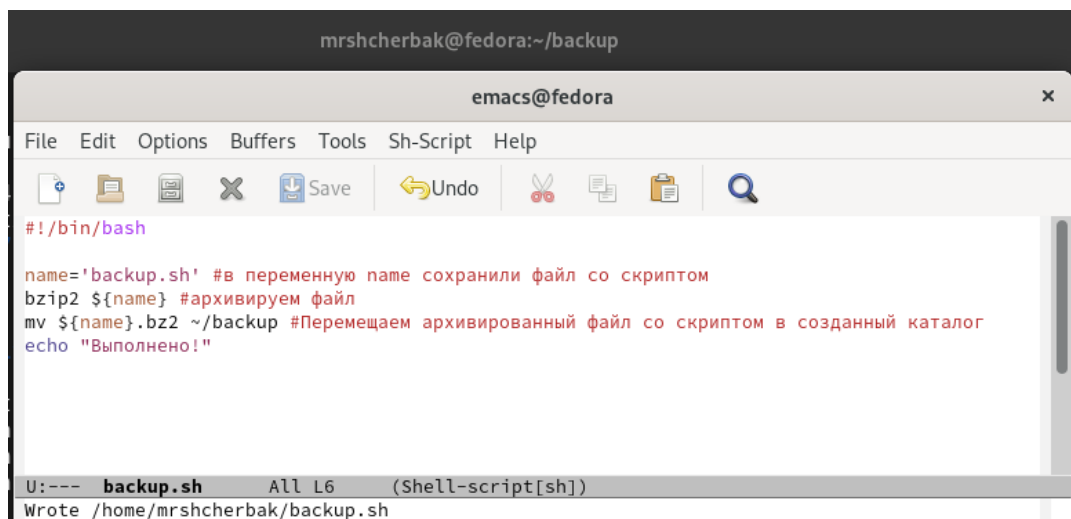


Рис. 3.6: Скрипт 1го задания

```
[mrshcherbak@fedora ~]$ cd backup
[mrshcherbak@fedora backup]$ ls
backup.sh.bz2
[mrshcherbak@fedora backup]$ bunzip2 -c backup.sh.bz2
#!/bin/bash

name='backup.sh' #в переменную name сохранили файл со скриптом
bzip2 ${name} #архивируем файл
mv ${name}.bz2 ~/backup #Перемещаем архивированный файл со скриптом в созданный каталог
echo "Выполнено!"
[mrshcherbak@fedora backup]$
```

Рис. 3.7: Просмотр содержимого архива

3. **Второе задание.** Создала файл skript2.sh, в котором писала второй скрипт, и открыла его в редакторе emacs. (Рис. 3.8 - Рис. 3.9).

- Предоставила право доступа на выполнение файлу skript2.sh
- Проверила файл на исполнение


```
mrshcherbak@fedora:~  
[mrshcherbak@fedora ~]$ touch skript2.sh  
[mrshcherbak@fedora ~]$ emacs &  
[1] 3302  
[mrshcherbak@fedora ~]$ chmod +x skript2.sh  
[mrshcherbak@fedora ~]$ ./skript2.sh  
Введённый аргументы  
[mrshcherbak@fedora ~]$ ./skript2.sh 0 1 2 3 4 5 6 7  
Введённые аргументы  
0  
1  
2  
3  
4  
5  
6  
7  
[mrshcherbak@fedora ~]$ ./skript2.sh 0 1 2 3 4 5 6 7 8 9 10 11 12  
Введённые аргументы  
0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
[mrshcherbak@fedora ~]$
```

Рис. 3.8: Создание файла и проверка на работоспособность

Скрипт последовательно печатает значения всех переданных аргументов.

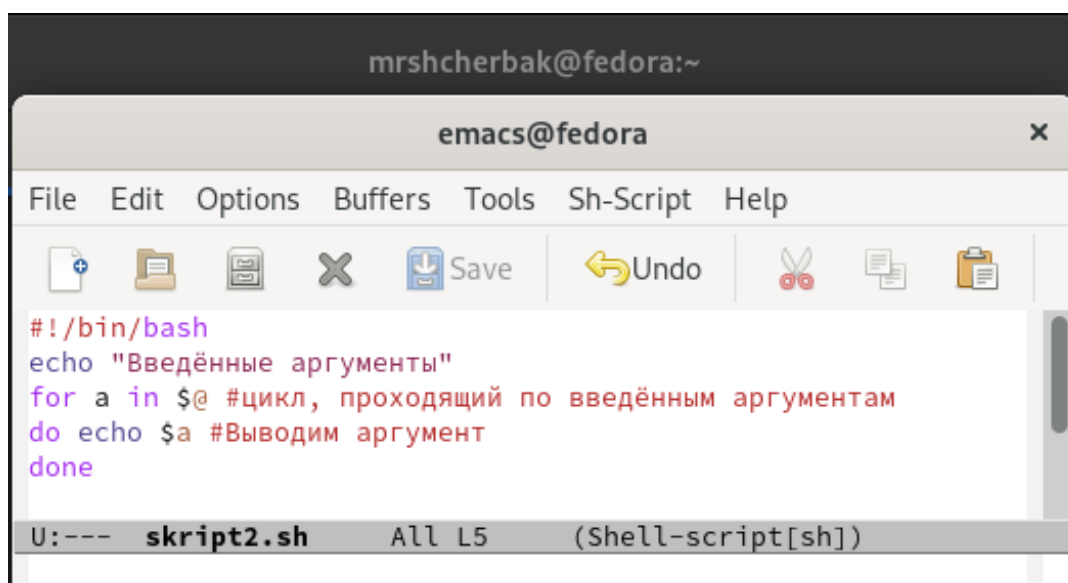


Рис. 3.9: Скрипт 2го задания

4. **Третье задание.** Создала файл skript3.sh, в котором писала третий скрипт, и открыла его в редакторе emacs (C-x C-s). (Рис. 3.10 - Рис. 3.12).
- Предоставила право доступа на выполнение файлу skript3.sh (chmod +x skript3.sh)
 - Проверила файл на исполнение

```
#!/bin/bash
a="$1" #В переменную а сохранили путь до заданного каталога
for i in ${a}/* #Цикл, проходящий по всем директориям и файлам
do
    echo "$i"
    if [[ -f $i ]]
    then echo "Файл"
    fi

    if [[ -d $i ]]
    then echo "Каталог"
    fi

    if [[ -r $i ]]
    then echo "Чтение разрешено"
    fi

    if [[ -w $i ]]
    then echo "Запись разрешена"
    fi

    if [[ -x $i ]]
    then echo "Выполнение разрешено"
    fi
done
U:--- skript3.sh All L6 (Shell-script[sh])
```

Рис. 3.10: Скрипт 3го задания

```
mrshcherbak@fedora:~  
[mrshcherbak@fedora ~]$ ./skript3.sh ~  
/home/mrshcherbak/abc1  
Файл  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/mrshcherbak/acd.txt  
Файл  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/mrshcherbak/adc.cpp  
Файл  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/mrshcherbak/adc.txt  
Файл  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/mrshcherbak/australia  
Каталог  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/mrshcherbak/backup  
Каталог
```

Рис. 3.11: Выполнение скрипта

Файл выдаёт информацию о нужном каталоге и выводит информацию о возможностях доступа к файлам этого каталога.

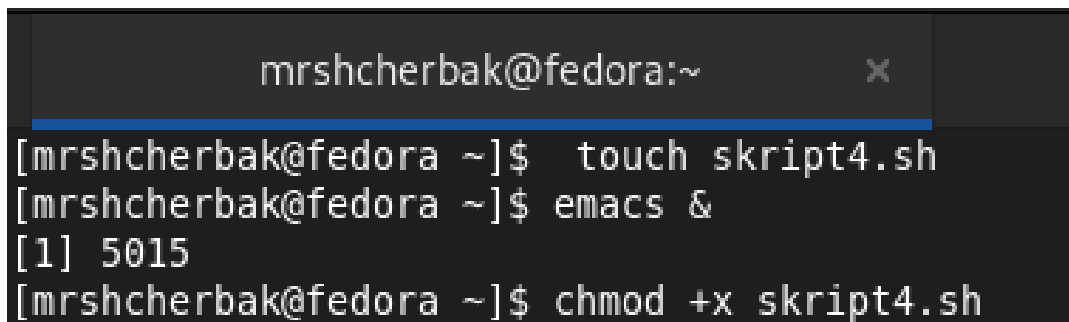
```
mrshcherbak@fedora:~  
/home/mrshcherbak/Загрузки  
Каталог  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/mrshcherbak/Изображения  
Каталог  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/mrshcherbak/Музыка  
Каталог  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/mrshcherbak/Общедоступные  
Каталог  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/mrshcherbak/Рабочий стол  
Каталог  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
/home/mrshcherbak/Шаблоны  
Каталог  
Чтение разрешено  
Запись разрешена  
Выполнение разрешено  
[mrshcherbak@fedora ~]$
```

Рис. 3.12: Результат

5. **Четвёртое задание.** Создала файл `skript4.sh`, в котором писала четвёртый скрипт, и открыла его в редакторе `emacs (C-x C-s)`. (Рис. 3.13 - Рис. 3.16).
- Предоставила право доступа на выполнение файлу `skript4.sh` (`chmod +x`

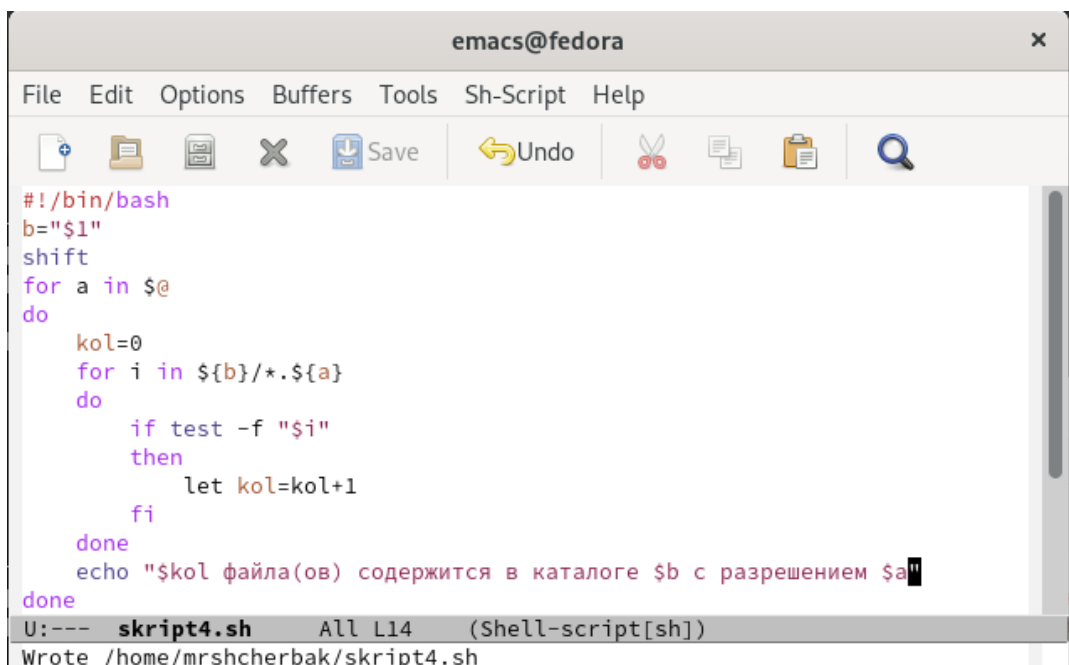
skript4.sh)

- Проверила файл на исполнение



```
mrshcherbak@fedora:~  
[mrshcherbak@fedora ~]$ touch skript4.sh  
[mrshcherbak@fedora ~]$ emacs &  
[1] 5015  
[mrshcherbak@fedora ~]$ chmod +x skript4.sh
```

Рис. 3.13: Создание файла и его право доступа на выполнение



```
emacs@fedora  
File Edit Options Buffers Tools Sh-Script Help  
#!/bin/bash  
b="$1"  
shift  
for a in $@  
do  
    kol=0  
    for i in ${b}/*.${a}  
    do  
        if test -f "$i"  
        then  
            let kol=kol+1  
        fi  
    done  
    echo "$kol файла(ов) содержится в каталоге $b с разрешением $a"  
done  
U:--- skript4.sh All L14 (Shell-script[sh])  
Wrote /home/mrshcherbak/skript4.sh
```

Рис. 3.14: Скрипт 4го задания

Файл получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет количество таких файлов в указанной директории.

```

[mrshcherbak@fedora ~]$ ./skript4.sh ~ pdf sh txt doc
0 файла(ов) содержится в каталоге /home/mrshcherbak с разрешением pdf
4 файла(ов) содержится в каталоге /home/mrshcherbak с разрешением sh
11 файла(ов) содержится в каталоге /home/mrshcherbak с разрешением txt
0 файла(ов) содержится в каталоге /home/mrshcherbak с разрешением doc
[mrshcherbak@fedora ~]$ touch exp.pdf
[mrshcherbak@fedora ~]$ touch exp.doc
[mrshcherbak@fedora ~]$ ./skript4.sh ~ pdf sh txt doc
1 файла(ов) содержится в каталоге /home/mrshcherbak с разрешением pdf
4 файла(ов) содержится в каталоге /home/mrshcherbak с разрешением sh
11 файла(ов) содержится в каталоге /home/mrshcherbak с разрешением txt
1 файла(ов) содержится в каталоге /home/mrshcherbak с разрешением doc
[mrshcherbak@fedora ~]$ ls
abcl      backup    exp.pdf   file12.txt  file14.txt-  may        reports    skript3.sh-  vvvvvvvvvv  Загрузки    Шаблоны
acd.txt   backup.sh-  feathers  file12.txt-  file.txt    monthly    ski_places  skript4.sh-  week3.txt   Изображения
adc.cpp   bin        '#file11.txt#'  file13.txt  Lab007      my_os      skript2.sh  skript4.sh-  work        Музыка
adc.txt   conf.txt   file11.txt  file13.txt-  lab07.sh    otchet.txt skript2.sh-  text.cpp     Видео       Общедоступные
australia exp.doc    file11.txt-  file14.txt  lab07.sh-   play       skript3.sh  text.txt     Документы   'Рабочий стол'
[mrshcherbak@fedora ~]$

```

Рис. 3.15: Результат

```

[mrshcherbak@fedora ~]$ touch exp.jpg
[mrshcherbak@fedora ~]$ ./skript4.sh ~ pdf sh txt doc jpg
1 файла(ов) содержится в каталоге /home/mrshcherbak с разрешением pdf
4 файла(ов) содержится в каталоге /home/mrshcherbak с разрешением sh
11 файла(ов) содержится в каталоге /home/mrshcherbak с разрешением txt
1 файла(ов) содержится в каталоге /home/mrshcherbak с разрешением doc
1 файла(ов) содержится в каталоге /home/mrshcherbak с разрешением jpg
[mrshcherbak@fedora ~]$ touch exp.svg
[mrshcherbak@fedora ~]$ touch exp.png
[mrshcherbak@fedora ~]$ ./skript4.sh ~ pdf sh txt doc jpg svg png
1 файла(ов) содержится в каталоге /home/mrshcherbak с разрешением pdf
4 файла(ов) содержится в каталоге /home/mrshcherbak с разрешением sh
11 файла(ов) содержится в каталоге /home/mrshcherbak с разрешением txt
1 файла(ов) содержится в каталоге /home/mrshcherbak с разрешением doc
1 файла(ов) содержится в каталоге /home/mrshcherbak с разрешением jpg
1 файла(ов) содержится в каталоге /home/mrshcherbak с разрешением svg
1 файла(ов) содержится в каталоге /home/mrshcherbak с разрешением png
[mrshcherbak@fedora ~]$

```

Рис. 3.16: Результат

Контрольные вопросы:

1. Командный процессор (командная оболочка, интерпретатор команд shell) – это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера.

В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек:

- оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;
- C-оболочка (или csh) — надстройка на оболочкой Борна, использующая Сподобный синтаксис команд с возможностью сохранения истории выполнения команд;

- оболочка Корна (или ksh)— напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна;

- BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation).

2. POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux-подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна.

3. Командный процессор bash обеспечивает возможность использования переменных типа строка символов. Имена переменных могут быть выбраны пользователем. Пользователь имеет возможность присвоить переменной значение некоторой строки символов. Например, команда «mark=/usr/andy/bin» присваивает значение строки символов /usr/andy/bin переменной mark типа строка символов. Значение, присвоенное некоторой переменной, может быть впоследствии использовано. Для этого в соответствующем месте командной строки должно быть употреблено имя этой переменной, которому предшествует метасимвол \$. Например, команда «mv afile{mark}» переместит файл afile из текущего каталога в каталог с абсолютным полным именем /usr/andy/bin. Оболочка bash позволяет работать с массивами. Для создания массива используется команда setc флагом -A. За флагом следует имя переменной, а затем список значений, разделённых пробелами. Например, «set -A states Delaware Michigan “New Jersey”». Далее можно сделать добавление в массив, например, states[49]=Alaska. Индексация массивов начинается с нулевого элемента.

4. Команда let является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение – это

единичный терм (term), обычно целочисленный. Команда `let` берет два операнда и присваивает их переменной. Команда `read` позволяет читать значения переменных со стандартного ввода: «`echo "Please enter Month and Day of Birth ?"`» «`read mon day trash`». В переменные `mon` и `day` будут считаны соответствующие значения, введенные с клавиатуры, а переменная `trash` нужна для того, чтобы отобразить всю избыточно введенную информацию и игнорировать её.

5. В языке программирования `bash` можно применять такие арифметические операции как сложение (+), вычитание (-), умножение (*), целочисленное деление (/) и целочисленный остаток от деления (%).

6. В (()) записывают условия оболочки `bash`, а также внутри двойных скобок можно вычислять арифметические выражения и возвращать результат.

7. Стандартные переменные:

- `PATH`: значением данной переменной является список каталогов, в которых командный процессор осуществляет поиск программы или команды, указанной в командной строке, в том случае, если указанное имя программы или команды не содержит ни одного символа /. Если имя команды содержит хотя бы один символ /, то последовательность поиска, предписываемая значением переменной `PATH`, нарушается. В этом случае в зависимости от того, является имя команды абсолютным или относительным, поиск начинается соответственно от корневого или текущего каталога.
- `PS1` и `PS2`: эти переменные предназначены для отображения промптера командного процессора. `PS1` – это промптер командного процессора, по умолчанию его значение равно символу \$ или #. Если какая-то интерактивная программа, запущенная командным процессором, требует ввода, то используется промптер `PS2`. Он по умолчанию имеет значение символа >.
- `HOME` — имя домашнего каталога пользователя. Если команда `cd` вводится без аргументов, то происходит переход в каталог, указанный в этой переменной.
- `IFS` — последовательность символов, являющихся разделителями в командной строке, например, пробел, табуляция и перевод строки (new line).

- MAIL — командный процессор каждый раз перед выводом на экран промптера проверяет содержимое файла, имя которого указано в этой переменной, и если содержимое этого файла изменилось с момента последнего ввода из него, то перед тем как вывести на терминал промптер, командный процессор выводит на терминал сообщение *You have mail* (у Вас есть почта).

- TERM — тип используемого терминала.

- LOGNAME — содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему.

8. Такие символы, как ' < > ? | " &, являются метасимволами и имеют для командного процессора специальный смысл.

9. Снятие специального смысла с метасимвола называется экранированием мета символа. Экранирование может быть осуществлено с помощью предшествующего мета символу символа `\`, который, в свою очередь, является мета символом. Для экранирования группы метасимволов нужно заключить её в одинарные кавычки. Строка, заключённая в двойные кавычки, экранирует все метасимволы, кроме `$`, `'`, `,`, `"`. Например, `-echo*` выведет на экран символ `*`, `-echoab'|'cd` выведет на экран строку `ab|cd`.

10. Последовательность команд может быть помещена в текстовый файл. Такой файл называется командным. Далее этот файл можно выполнить по команде: `«bash командный_файл [аргументы]»`. Чтобы не вводить каждый раз последовательности символов `bash`, необходимо изменить код защиты этого командного файла, обеспечив доступ к этому файлу по выполнению. Это может быть сделано с помощью команды `«chmod +x имя_файла»`. Теперь можно вызывать свой командный файл на выполнение, просто вводя его имя с терминала так, как будто он является выполняемой программой. Командный процессор распознает, что в Вашем файле на самом деле хранится не выполняемая программа, а программа, написанная на языке программирования оболочки, и осуществить её интерпретацию.

11. Группу команд можно объединить в функцию. Для этого существует ключевое слово `function`, после которого следует имя функции и список команд, заключённых в

- фигурные скобки. Удалить функцию можно с помощью команды `unset` с флагом `-f`.
12. Чтобы выяснить, является ли файл каталогом или обычным файлом, необходимо воспользоваться командами `test -f [путь до файла]` (для проверки, является ли обычным файлом) и `test -d [путь до файла]` (для проверки, является ли каталогом).
13. «`set`» можно использовать для вывода списка переменных окружения. В системах *Ubuntu* и *Debian* команда «`set`» также выведет список функций командной оболочки после списка переменных командной оболочки. Поэтому для ознакомления со всеми элементами списка переменных окружения при работе с данными системами рекомендуется использовать команду «`set | more`». Команда «`typeset`» предназначена для наложения ограничений на переменные. Команду «`unset`» следует использовать для удаления переменной из окружения командной оболочки.
14. При вызове командного файла на выполнение параметры ему могут быть переданы точно таким же образом, как и выполняемой программе. С точки зрения командного файла эти параметры являются позиционными. Символ `$` является метасимволом командного процессора.
15. Специальные переменные:
- `$` –отображается вся командная строка или параметры оболочки;
 - `$?` –код завершения последней выполненной команды;
 - `$ (2 Таких знака)` –уникальный идентификатор процесса, в рамках которого выполняется командный процессор;
 - `$!` –номер процесса, в рамках которого выполняется последняя вызванная на выполнение в командном режиме команда;
 - `$` значение флагов командного процессора;
 - `${#}` –возвращает целое число –количество слов, которые были результатом `$`;
 - `${#name}` –возвращает целое значение длины строки в переменной `name`;
 - `${name[n]}` –обращение к `n`-му элементу массива;
 - `${name[*]}` –перечисляет все элементы массива, разделённые пробелом;
 - `${name[@]}` –то же самое, но позволяет учитывать символы пробелы в самих переменных;

- `${name:-value}` –если значение переменной `name` не определено, то оно будет заменено на указанное `value`;
- `${name:value}` –проверяется факт существования переменной;
- `${name=value}` –если `name` не определено, то ему присваивается значение `value`;
- `${name?value}` –останавливает выполнение, если имя переменной не определено, и выводит `value` как сообщение об ошибке;
- `${name+value}` –это выражение работает противоположно `${name-value}`. Если переменная определена, то подставляется `value`;
- `${name#pattern}` –представляет значение переменной `name` с удалённым самым коротким левым образцом (`pattern`);
- `${#name[*]}` и `${#name[@]}` – эти выражения возвращают количество элементов в массиве `name`.

4 Выводы

Таким образом, в ходе ЛРН^{№10} я изучила основы программирования в оболочке ОС UNIX/Linux. Научилась писать небольшие командные файлы.